EXPRESS MAIL NO. EV301186335US

SYSTEM AND ASSOCIATED METHODS FOR SOFTWARE ASSEMBLY

Cross-Reference to Related Application.

[0001] This application claims priority under 35 U.S.C. §119 from UK patent application number 0314800.4, filed June 25, 2003.

Field of the Invention.

[0002] This invention relates to a system and associated methods for software assembly.

Description of Related Arts.

[0003] It will be appreciated that an application is required to cause a processing apparatus, which may be a computer, to perform a task. This application may be defined in hardware or software. It is convenient to consider software applications since these are perhaps more common.

[0004] Such software applications have traditionally followed what may be thought of as two paths: off the shelf packages and bespoke packages. Each path has its own advantages and disadvantages.

[0005] For example, off the shelf applications are perhaps cheaper than bespoke applications but generally cannot be tailored to a user's needs as much as a bespoke application. On the other hand since bespoke applications are written specifically to a user's requirements they may be tailored precisely to requirements but can be costly. The cost of a bespoke application of course includes the development, debugging and maintenance costs. Bespoke applications can often be error prone since they are written specifically for user's needs and therefore they are unlikely to benefit from the extensive testing that an off the shelf application will experience.

[0006] Therefore, a user wishing to perform a task must make the choice of whether to use an off the shelf application or to use a bespoke application. The off the shelf application may not provide all of the features that they require but may be cheaper and less prone to error. The bespoke application may be much more tailored to the user's needs but is likely to be slow to develop and costly to develop and maintain.

[0007] Many prior art systems are based around object oriented programming languages such as C#, C^{++}, JAVA, and the like. Such languages define objects, or data structures, that have predetermined and rigid data formats. If such an object receives data that is not in the required format then an error will be generated and the program may fail. Such well formulated data structures are not necessarily well suited to data generated in the real world which does not necessarily follow such well defined data formats.

[0008] In the prior art attempts have been made to solve this problem and an example of a prior art document that shows a method and system for implementing process-based web applications is shown in EP 1 187 010. However, such solutions are limited in their applicability and may not increase the speed of development and robustness as much as may be desired.

Summary of the Present Invention.

[0009] According to a first aspect of the invention there is provided a method of creating an application comprising:

creating a plurality of nodes each node being capable of receiving data, processing that data within the node according to a set of predetermined rules and making an output;

creating a specification that defines how the nodes interact with one another and defines resources useable by the nodes; and

providing a run time environment that interconnects the nodes according to the specification such that data input to the application is processed by

one or more of the nodes and if further processing is required, forwarded to other nodes for that further processing.

[0010] An advantage of such a method is that it allows an application to be implemented much more rapidly than prior art, particularly bespoke, applications. Such rapid development saves both cost and time for a party that is developing the application.

[0011] The method may comprise selecting one or more of the nodes from a library of nodes. Such an arrangement is convenient because, again, it speeds development of the application; if a node can be re-used without having to be re-written then the development time will be reduced.

[0012] Preferably each node is defined in software. The skilled person will of course appreciate that the nodes could also be implemented in hardware.

[0013] Each node preferably comprises a plurality of layers with each layer being arranged to perform a predetermined function. In the preferred embodiment each node comprises three layers. However, in additional or alternative embodiments any number of layers may be provided. For example, 1,2,4,5,6,7,8,9,10, 15 or more layers may be provided.

[0014] Conveniently the layers of a node are arranged to be interchangeable such that altering one or more of the layers can change the overall functionality of a node. Conveniently, a layer may be selected from a library of layers. Such an arrangement is convenient because again the amount of work in developing the application can be significantly reduced.

[0015] The method may comprise arranging one of the layers of the node to act as a transport layer arranged to receive and send data to and from the node. Such a layer is advantageous since is allows a node to be readily tailored to work

with a variety of different transport mechanisms. For example, a transport layer may be provided that interfaces with TCP/IP, HTTP, SMTP, SOAP or the like communications.

[0016] Further, the method may arrange that one of the layers of the node acts as a message transceiver arranged to send and receive messages to other nodes to which that node is connected. The message transceiver layer of the node may be arranged to discover the identity of nodes to which it is connected at runtime of the application.

[0017] The method may arrange that one of the layers of the node acts as a rule processing engine arranged to apply predetermined rules to data that the node receives.

[0018] Preferably, the rule processing engine layer of a node uses forward chaining rule logic. Such a method is particularly convenient because is allows data received by the rule processing engine layer to fire one or more rules which may reduce, perhaps significantly, the amount of storage required for the rules; the use of forward chaining rules may significantly reduce the number of rules required in order to describe a situation and will reduce processing power requirements since only rules that are required are fired; irrelevant rules are not generally evaluated. Reducing the amount of storage in this manner provides a technical advantage and may also make the application faster; if there are less rules to process it is likely that the processing speed can be increased.

[0019] The method may provide the rules (e.g. a rule set) that are used by the rule processing engine in a file which will generally be a specific file for each rule set. Such a method is convenient because it allows the rules to be readily amended and also it allows the rules to be reused by a plurality of rule processing engines. Conveniently, the file in which the rules are located is specified by a link, sometimes referred to as a pointer.

[0020] Each rule set may be conveniently be thought of as an asset that a node can utilise.

[0021] Further, the method may define predetermined messages that can be passed between the nodes. These messages may also be thought of as an asset that the node can utilise.

[0022] Conveniently, the specification defines the messages and/or rule sets that are to be used by the application. Such an arrangement is convenient because it allows the same message and/or rule set to be re-used and can therefore reduce, perhaps significantly, the complexity of the files required for the application.

[0023] The method may arrange for the messages to be in XML.

[0024] Messages may be processed by the rules within the node. The rules may be arranged such that they are only triggered if the message contains predetermined trigger data. An advantage of such an arrangement is that the system may become more robust; the format of the messages does not matter and the message is simply processed to determine whether it has the relevant trigger data to trigger a rule. This is in contrast to prior art systems, may be those based upon object oriented techniques, that are likely to fail if the data received is not in the predetermined format.

[0025] Conveniently, the layers of the node are specified by files and generally each layer is specified by a specific file.

[0026] Preferably the or each node in the application is provided within a pattern that defines how nodes therein interact with one another. Such a method

is convenient because again, it can reduce the amount of work in creating the application.

[0027] The method may provide a library of patterns that can be used in creating an application. Patterns may be provided to perform tasks that are commonly required. As an example a pattern may be provided that performs any of the following: receives and processes an invoice; receives and processes an email; raises an invoice; adds an entry to a database (perhaps using SQL or the like).

[0028] The specification that may be thought of as a blueprint. The specification may determine any of the following non-exhaustive list: which nodes are to be used; which nodes interact with one another; which patterns are to be used, which assets are to be used.

[0029] The method may use the specification to deploy files that are used to define the application specified therein. An advantage of this is that additional effort (such as compilation, assembling, etc.) is not required in order to provide a working application. Once components of the application have been provided they are run in the run time environment and in such embodiments there is no need to provide compilers, etc. in order to generate code for execution. Therefore, such methods should be less prone to error, allow easier maintenance of the application, be quicker to develop, etc.

[0030] The files specifying the application may be XML files. XML files are convenient because XML is becoming widely available and as such is becoming readily understood. Further, the XML files may be able to more readily receive and process XML data which is becoming more common. However, the method may comprise deploying nodes and/or node layers that can receive and process data that is non XML.

[0031] Conveniently, the method provides a graphical tool that allows a user to specify the application, including specifying the specification. Such a method may increase the ease of use of the system. The skilled person will appreciate the proliferation and acceptance of Graphical User Interfaces which may be due to their ease of use. A graphical tool may be readily deployable on such GUI's and also more convenient than for example a command line for a user to use.

[0032] The graphical tool may allow a user to select components from a library. The components may comprise any of the following non-exhaustive list: nodes; node layers; specification; patterns; messages; rule sets; style sheets (which may be XML); schemas (which may be XML). As discussed above, the messages and rule sets may be thought of as assets of the node. Further, the style sheets and schemas may also be thought of as assets of a node.

[0033] Further, the graphical tool may allow a user to specify how the patterns and/or nodes interact with one another.

[0034] The specification may be created and/or edited by the graphical tool.

[0035] Further, the graphical tool may allow a user to edit or otherwise manipulate the rule sets and/or other assets.

[0036] Conveniently, the method provides a run time environment that allows the specification to be processed to provide the application. The specification may contain any of the following non-exhaustive list: nodes, patterns, assets. Preferably, the run time environment processes the files created and/or deployed by the method. Such an environment is advantageous because it may provide for faster modification of the application. For example, modification may be made to one or more of the run time files which can then be used by the run time environment to produce the modified application. Prior art solutions would generally need recompiling, etc. in order to change the application. It is

therefore likely that the method is generally quicker to modify the application than the prior art.

[0037] According to a second aspect of the invention there is provided a computer system arranged to create an application; said system comprising a node creation means arranged to create a definition specifying one or more nodes, each node being capable of processing data according to a set of predetermined rules and generating an output therefrom, the system further comprising a linking means capable of connecting one or more nodes such that data can pass between the nodes and being arranged to interact with the node creation means to modify the definition and a deployment means arranged to deploy the application from the definition created by the node creation means and the linking means, wherein the deployment means deploys the application according to a specification defined by a specification means.

[0038] An advantage of such a system is that it may provide for faster development of applications than prior art arrangements.

[0039] The system will generally comprise one or more processors capable of processing the definition deployed by the node creation means and the linking means.

[0040] The linking means may be arranged to deploy the definition allowing nodes running on the same processor(s) within a single processing apparatus.

[0041] The computer system may comprise a plurality of processing apparatus, each remote from the other, and having a connection therebetween capable of transmitting data between the processing apparatus. In such an arrangement the linking means may be arranged to deploy the definition allowing nodes provided by the definition running on processors in processing apparatus remote from one another. Such an arrangement is convenient since it makes the

computer system scaleable and simplifies the running of an application across a number of pieces of processing apparatus.

[0042] Conveniently the deployment means deploys the definition that causes the nodes to communicate with one another using HTTP, direct memory protocols or other suitable protocols. Such an arrangement is convenient because it allows the nodes to communicate using standard protocols, perhaps making them widely useable and understood.

[0043] The node creation means may be arranged to utilise predetermined and/or pre-written definitions in order to reduce the time and effort required to create the application. The pre-written definition may be provided in one or more libraries. This will also generally be less error prone and provides better integrity and future maintenance when compared to causing the node creation means to create the definition for each of the nodes.

[0044] The computer system may comprise a pattern creation means arranged to create patterns of nodes. Such patterns may make it convenient for a user to create the application since it may reduce the work involved in creating the nodes. For example, if a pattern of a plurality of nodes is repeated within the application then by repeating the pattern the work involved in creating each of the nodes within the pattern is likely to be reduced.

[0045] Conveniently a pattern cloning means arranged to clone a pattern of nodes is provided. Such an arrangement may speed the creation of the application.

[0046] Preferably, the computer system comprises a rule creation means arranged to allow the predetermined rules to be created and/or edited. Such an arrangement is convenient because it allows the functionality of a node to be

altered since it is the rules that govern how data received by the node is processed.

[0047] The computer system conveniently comprises one or more of the following: a node storage means; a pattern storage means; a rule storage means.

[0048] The second aspect of the invention may have any of the features discussed in relation to the first aspect of the invention.

[0049] According to a third aspect of the invention there is provided a machine readable medium containing instructions which when read onto a computer cause that computer to perform the method of the first aspect of the invention.

[0050] According to a fourth aspect of the invention there is provided a machine readable medium containing instructions which when read onto a computer cause that computer to function as the computer according to the second aspect of the invention.

[0051] According to a fifth aspect of the invention there is provided a data-structure comprising a plurality of XML files and in particular comprising a specification file and a plurality of asset files the asset files being capable of interacting with one another and the specification file determining how the asset files interact with one another to provide an application having a predetermined functionality.

[0052] According to a sixth aspect of the invention there is provided a machine readable medium containing instructions that provide the data structure according to the fifth aspect of the invention.

[0053] According to a seventh aspect of the invention there is provided a method of implementing an application comprising:

creating a plurality of nodes each node being capable of receiving data, processing that data according to a set of predetermined rules and making an output;

interconnecting the plurality of nodes such that data input to the application is processed by one or more of the nodes and if further processing is required, forwarded to other nodes for that further processing.

[0054] According to an eighth aspect of the invention there is provided a method of creating an application, generally defined in software, the method comprising providing an XML file defining a specification and one or more assets used by the specification, providing one or more nodes each defined by XML and which interact with each other and/or the assets according to the specification in order to process data to provide the application.

[0055] The nodes may be provided by separate XML files. However, in alternative embodiments the nodes may be defined by the file providing the specification. It is generally desirable to provide them as a separate file to avoid the creation of lengthy files that are cumbersome to amend and upkeep.

[0056] The features discussed in relation to any other aspects of the invention may be applicable to the eighth aspects of the invention.

[0057] According to a ninth aspect of the invention there is provided a machine readable medium containing instructions that provide the data structure according to the fifth aspect of the invention.

[0058] According to a tenth aspect of the invention there is provided a computer system comprising a memory and processing means arranged to

process instructions held in the memory wherein the memory is arranged to hold one or more files containing data, generally in XML format, at least one of the or each files defining a specification, further, at least one of the files defining a node capable of processing data and the specification being arranged to define how that node should process data.

[0059] According to an eleventh aspect of the invention there is provided a processing apparatus having a processor and memory the memory containing both program code and data wherein the program code and the data are substantially written in the same language.

[0060] Such an apparatus is advantageous because it may provide the ability to develop new applications using less program code than prior art systems.

[0061] The memory may hold presentation data specifying how output from the apparatus should be presented, the presentation data being held in the same format as the program code and the data.

[0062] In a preferred embodiment the program code and the data may be written in XML.

[0063] The processor may have a kernel running thereon which is preferable written in a platform independent language. Such a platform independent language is preferred since it will allow the kernel to be readily ported to a number of different platforms. In one embodiment the kernel is written in Java and a Java Virtual Machine (JVM) is provided on the apparatus in order to run the kernel.

[0064] The machine readable medium of any of the aspects of the invention may be any one or more of the following: a floppy disk; a CDROM/RAM; a DVD ROM /RAM (including +R/+RW,-R/-RW); any form of magneto optical

disk; a hard drive; a memory; a transmitted signal (including an internet download, file transfer, or the like); a wire; or any other form of medium.

Brief Description of the Drawings.

[0065] Embodiments of the invention will now be described by way of example only and with reference to the figures of which:

[0066] **Figure 1** shows a computer system arranged to carry out one or more embodiments of the present invention;

[0067] **Figure 2** shows detail of the memory of the computer system of Figure 1;

[0068] **Figure 3** shows a conceptual overview of one embodiment of the present invention;

[0069] **Figure 4** shows a conceptual representation of a node according to one embodiment of the present invention;

[0070] **Figure 5** shows a conceptual representation of a pattern of nodes according to one embodiment of the present invention;

[0071] **Figure 6** shows an example of a situation to which an embodiment of the present invention may be applied;

[0072] **Figure 7** shows a flowchart giving an overview of one embodiment of the present invention;

[0073] **Figure 8 to 30** show screenshots that may be displayed by the computer system of Figure 1; and

[0074] **Figure 31** shows two possible physical arrangements for a conceptual specification of an application.

Detailed Description of the Invention.

[0075] Figure 1 shows a computer 100 arranged to accept data and to process that data. The computer 100 comprises a display means 102, in this case a Cathode Ray Tube (CRT) display, a keyboard 104, a mouse 106 and processing circuitry 108. It will be appreciated that other display means such as LEP (Light Emitting Polymer), LCD (liquid crystal display), projectors, televisions and the like may be equally possible.

[0076] The processing circuitry 108 comprises a processing means 110, a hard drive 112 (containing a store of data), memory 114 (RAM and ROM), an I/O subsystem 116 and a display driver 117 which all communicate with one another, as is known in the art, via a system bus 118. The processing mean 110 typically comprises at least one INTEL™ PENTIUM™ series processor, (although it is of course possible for other processors to be used) and performs calculations on data. The other processors may include processors such as the AMD™ ATHLON™, POWERPC™, DIGITAL™ ALPHA™, and the like.

[0077] The hard drive 112 is used as mass storage for programs and other data. The memory 114 is described in greater detail below and with reference to Figure 2.

[0078] Other devices such as CDROMS, DVD ROMS, scanners, etc. could be coupled to the system bus 118 and allow for storage of data, communication with other computers over a network, etc.

[0079] The I/O (Input/Output) subsystem 116 is arranged to receive inputs from the keyboard 104 and from the processing means 110 and may allow communication from other external and/or internal devices. The display

driver 117 allows the processing means 110 do display information on the display 102.

[0080] The processing circuitry 108 further comprises a transmitting/receiving means 120, which is arranged to allow the processing circuitry 108 to communicate with a network. The transmitting/receiving means 120 also communicates with the processing circuitry 108 via the bus 118.

[0081] The processing circuitry 108 could have the architecture known as a PC, originally based on the IBM™ specification, but could equally have other architectures. The server may be an APPLE™, or may be a RISC system, and may run a variety of operating systems (perhaps HP-UX, LINUX, UNIX, MICROSOFT™ NT, AIX™, or the like). The processing circuitry 108 may also be provided by devices such as Personal Digital Assistants (PDA's), mainframes, telephones, televisions, watches or the like.

[0082] Figure 2 shows the memory 114 of the computer 100 of Figure 1 in greater detail. It will be appreciated that although reference is made to a memory 114 it is possible that the memory could be provided by a variety of devices. For example, the memory may be provided by a cache memory, a RAM memory, a local mass storage device such as the hard disk 112, any of these connected to the processing circuitry 108 over a network connection such as via the transmitting/receiving means 120. However, the processing means 110 can access the memory via the system bus 118 to access program code to instruct it what steps to perform and also to access the data samples. The processing means 110 then processes the data samples as outlined by the program code.

[0083] The memory 114 is used to hold instructions that are being executed, such as program code, etc., and contains a program storage portion 150 allocated to program storage. The program storage portion 150 is used to hold program

code that can be used to cause the processing means 110 to perform predetermined actions.

[0084] The memory 114 also comprises a data portion 152 allocated to holding data and in embodiments of the present invention in particular provides a Node storage means 202, Pattern storage means 204, Asset storage means 206, and a Rule storage means 208. The function of these will be expanded upon hereinafter.

[0085] In this embodiment, the program code stored in the data portion 152 includes a Node Creation means 212, Pattern Creation means 214, an Asset Creation means 216, a Rule Creation means 218, a Pattern Cloning means 220, a Linking means 222 and a Deployment means 224. Again, the function of these will be expanded upon hereinafter.

[0086] Figure 3 gives an overview of the one aspect of the present invention. Implementation of a system is divided into two stages - the design stage and the deployment stage. In the design stage, the system developer develops an application specification which may be thought of as a blueprint 300. This comprises information relating to the format of data that will be accepted by the system (data formats 302), the expected processes to be performed in terms of the patterns of interaction between software and/or hardware elements of the system (patterns 303) and the rules governing those processes (rules 304).

[0087] The deployment stage is effectively the specification 300 in action, i.e. the specification 300 in use in the real world. In deployment, the specification 300 is applied to a run time environment, which in this example comprises the programming language such as Java, data representation of text and communicating using protocols such as http and XML. The skilled person will be familiar with these languages, which can be noted for their suitability for use in conjunction with the Internet and their cross platform usability (i.e. their

ability to run on a variety of different computers and operating systems). Equally, the skilled person will be familiar with the concept of a platform being the underlining software and/or hardware for a system.

[0088] Figure 4 shows an example of a Node 400 which is defined by a definition which may have been generated by the node creation means. One or more and generally a plurality of such nodes are suitable for forming a system according to the invention. Nodes 400 are arranged to interact with each other using messages written in XML, which may be referred to as XML messages.

[0089] The code shown in the appendices is written in XML (eXtensible Mark-up Language). XML requires pairs of tags to be placed within a document. Theses tags do not specify how the information should be presented but specify the content of the information between the pairs of tags. The skilled person will fully understand XML, but a full description can be found at http://www.w3.org, and the brief description below will aid his/her understanding.

[0090] The skilled person will appreciate how an XML document is structured: written in words, or data sub-items, which are collected into data sub-item groups. The data sub-item groups can comprise sentences, paragraphs, or simply collections of words. The data sub-item groups, or even just data sub-items, are placed between pairs of tags.

[0091] The tags appear as follows: <variable>, and </variable>, with variable being any word, or character string acceptable according to the XML recommendation. Further, each data sub item group can be itself broken down into a number of sub-items. This structure is convenient and allows for easy manipulation and searching of the complete data item.

[0092] Each Node is provided by a logical definition. Properties of a node include an identifier, name, and description. Each Node has at least one

information receptor but may have more. Each receptor is dedicated to a particular type of information. By default each node has at least one receptor capable of receiving any XML information. Additional receptors can be used to process any non-XML information, including but not limited to standard, flat, information streams and binary data. Each node is defined by a separate XML file held in the memory 114 and in particular within the node storage means 202. As the skilled person will appreciate it is equally possible to specify components of the system in fragment of a file. The two extremes may be thought of specifying each component in a separate file and specify all of the components as a fragment within a single file. Embodiments of this invention may take any position between these two extremes. The components may comprise any of the following non-exhaustive list: specification; nodes; node layers; patterns; messages; rule sets; style sheets (which may be XML); schemas (which may be XML).

[0093] The receptors enable a node to receive information. After receiving information via the receptors, nodes can use information processing logic to process that information. The logic can take different forms. The incoming information is assumed to be XML by default. For XML information the default form of logic is context-sensitive rules that are activated by the information received. This logic is executed by a forward-chaining rules engine. The logic itself is represented in XML format. If the incoming information is not XML, then logic dedicated to the information format can be used by the node. This dedicated logic can be in the form of scripts or compiled Third Generation Language (3GL) programs such as JAVA. The context sensitive logic can be executed conditionally, based on the incoming information. Conditions may be based on XML tags, values or structures.

[0094] As the logic is executed, factual information is derived therefrom. This information is stored in working memory 112,114. If any part of this information is required beyond a single invocation of the node then it can be

placed in a cache. During logic execution, additional resources can be used. These resources can be in various formats, including XML, XSL, XSD, text, graphics, etc. For XML information, an extensible set of operations is available for the input, output and manipulation of XML information. Examples of these operations include, Read, Write, Transform, Copy, Service Call, etc.

[0095] The incoming information, the logic and the additional resources that are used during node execution are all defined separately and have no knowledge of the context in which they will be used. This enables the same information, logic and resources to be used with different nodes. The information, logic and resources are also location transparent and can be situated anywhere, so long as they are accessible using URL syntax. This also enables the transportation of information, the logic to process the information and any additional resources that are required over the Internet Protocol (HTTP), enable mobile information and logic with autonomous processing capabilities.

[0096] Within each node, and as described further below, messages received by the node are inspected and decisions made based on their content. Based on these decisions, other nodes 400 may be contacted. To accomplish this, each node 400 comprises (on a conceptual level) three layers as are now described. In other embodiments, each layer could comprise another number of layers. Each of these layers is represented by the one of the three concentric circles 402, 404, 406 of Figure 4.

[0097] The first, innermost, layer is a rule processing, or logic, engine 402 and comprises a 'forward chaining' logic engine. The engine uses XML rule sets to make decisions based on the content of the incoming XML messages. The decisions may include triggering messages or calls to other nodes or changing the content of the messages. A consequence of writing both the message and the rule sets in XML is that the system can be configured dynamically- i.e. there is no need to compile code every time a change is made to the system. This will be

expanded upon later. In situations where native XML is insufficient custom engines can be developed using means such as Java classes.

[0098] The second, middle, layer may be thought of as an agent 404 which acts as a message transceiver arranged to send and receive messages to other nodes to which this node is connected. The message transceiver 404 is arranged, respectively, to send and receive the XML messages to and from other message transceivers 404. Message transceivers 404 are capable of discovering the location of others at runtime when the nodes are executed in the run time environment, in a manner similar to hosts on a peer-to-peer network. As will be appreciated from Figure 4, the message transceiver 404 may be thought of as containing and/or using the rule processing engine 402. The message transceivers 404 can be deployed locally within a Java Virtual Machine (JVM) as native Java services or may be deployed using the third layer 406 as a means such as a web service. Thus, the message transceiver may be thought of as an intelligent software agent which is capable of reacting to the content of the incoming messages to be able to function. The rules are triggered by trigger data within the messages that are specific to the rule; if there is no trigger data within a message then none of the rules will trigger. However, the message transceiver will continue to function if no rules trigger and therefore, the message transceiver can accept messages in any format although no action may occur if the message does not contain any trigger data for the rules.

[0099] The third layer is a gateway 406, arranged to provide access to the node 400 over a variety of protocols (for example HTTP, Java and the like). This third layer may be thought of as a transport layer arranged to receive and send data to and from the node. A message transceiver 404 uses its associated gateway 406 to locate and send messages to and from other message transceivers 404. Further, external message transceivers 404 use the gateway 406 to locate the associated message transceiver 404.

[0100] The rule processing engine 402, the message transceiver 404 and the gateway 406 are separable from each other in that each of the layers (for example the rule processing engine 402) may be replaced by another version of that layer, causing the node 400 to work in a different way when compared to a node having a different combination of layers. Each layer is therefore chosen according to the intended purpose of the node 400. The node 400 may for example be arranged to process customer orders, it may be arranged to model a cell in a biological system, it may be arranged to provide a web based service, etc. The node 400 can be programmed to perform any functions and need only be provided with appropriate layers 402, 404, 406.

[0101] In this particular embodiment each of the rule processing engine 402, the message transceiver 404 and the gateway 406 are provided by separate XML files. The identities of these files is specified within the specification 300 such that run time environment can access the respective files when required. It will be appreciated that in other embodiments the components of a node may be merged into a single file or into other combinations. However, the location of the components would be specified within the specification 300 such that the run time environment could locate each component.

[0102] As is shown in Figure 5, nodes 400 are arranged in a pattern 500 to form all or part of a system. A pattern comprises one or more interacting nodes 400 with defined relationships between them. The nodes 400 in a pattern are capable of exchanging XML messages via communication links 502. Nodes 400 may be run on the same processing circuitry 108 and links 502 between such nodes 400 may be thought of as virtual links. Alternatively, or additionally, the nodes 400 may be run on different processing circuitry 108 and such links 502 may correspond to a connection between two processing circuits 108. The specification 300 may be used to determine the location at which the node will run.

[0103] Each node is autonomous and has no knowledge of its surroundings. During the execution of logic, if a node encounters the need to call another node, it refers to its Pattern which provides information as to where that node 400 is connected. A pattern defines which nodes can call other nodes and using which information receptors. The aim of a pattern is to encapsulate a set of interacting nodes that perform a higher-level function. A pattern can contain any number of nodes, depending on the application context.

[0104] The specification 300 that has been created defines patterns of interaction between message transceivers. The interaction is not specified and it is merely noted that it is likely to occur. To give an example, message transceivers may be provided to simulate a wall, a floor and a door. The patterns of interaction may be that the floor is attached to a bottom region of the walls. The pattern of interaction for the door is that it is likely that the door will be provided in a wall; the position of the door within the wall is not specified.

[0105] There are two types of patterns, logical and physical. The logical patterns define the nodes and their relationships. The physical patterns also define the information, logic and resources the nodes can use and the exact physical manifestation of each node. For example, a node may be represented by a script, a 3GL program or some other mechanism. The pattern definitions are also stored in XML format.

[0106] Pattern splicing enables multiple patterns to be joined together to perform yet higher level functions. The patterns are spliced by creating a link between two nodes residing in different patterns. This splicing information is contained in the specification 300.

[0107] A specification 300 provides context to the abstract definition of nodes and patterns. A specification 300 defines, categorises and constrains the way in which patterns can be spliced together. A specification 300 categorises

information, logic, resources and physical patterns. It also defines the architectural layers, which different categories of patterns must reside within as well as the splicing information for pattern connections. The specification 300 are also stored in XML format.

[0108] The nodes 400 process rules. In the deployment stage, the rules are held within the innermost layer: the rule processing engine 402. In the design stage, they are held in the Rule storage means 208 which acts as a repository. In this embodiment the rules follow a structure that is intuitive for a user to understand: <event>-<condition>-<action>. The rules are only fired if the <condition> is satisfied by the incoming <event> content, which may be thought of as trigger data, of the XML message received by the message transceiver 404 (which may be a business document or the like).

[0109] It will be appreciated that some rules will not be able to execute with the information that is currently available and may require other additional information before they can execute. For example, consider the rule: Only sell shares if the price is greater than £x. This rule cannot execute unless the price of those shares is known. A second rule is therefore required that acquires the price of the shares. The first rule references the second through use of the XPath language; a non-XML language that is used to identify parts of XML documents. Through use of the XPath references the first rule is triggered when the second rule is fired; i.e. in this example, when the share prices are available the first is triggered to determine whether the shares should be sold. The concept of forward-chaining rules in this manner will be well understood by the skilled person.

[0110] In summary of the above, each specification 300 therefore encapsulates, patterns spliced together. Patterns contain interacting nodes, which consume and process incoming information and additional resources. This constitutes a specification. A blueprint can therefore be perceived as a

specification consisting of a collection of native XML documents. In one embodiment XML documents are provided for the assets as defined in the specification which may comprise any of the following: a rule; a rule set; a message; a style sheet; a schema.

[0111] It may be efficient to define certain assets within the specification 300 rather than in a separate XML file (or portion of an XML file) if the asset is to be used by a plurality of other components (such as a node or a pattern, etc.). If a system comprise 50 000 nodes, each of which could require a common message, it would be efficient to define that message within the specification 300 rather than in a separate XML file (or a portion of an XML file).

[0112] A graphical tool is provided that allows a user to develop all of the components of the application. In this sense a component may be thought of as any of the following: a node, a layer of a node, a rule, a rule set, a message, a pattern, an asset, a schema, a style sheet, a specification. Once all of the components of the application have been created then they can be deployed using the run time environment. Unlike prior art systems and methods this deployment does not require extra processing such as compilation or assembly and the components are simply processed by the run time environment.

[0113] Deployment comprises causing the Deployment means 224 to access the components and process them accordingly. The deployed specification 300 can process data 310 provided to the system as an input and provide a result 312 as an output.

[0114] For the remainder of this description, and as only one example of the many possible applications for this system, an application relating to car manufacture is used. It should however be clearly understood that the present invention is not limited to the field of controlling business processes. It may, as in the examples given above, be used to monitor or model reactions in a nuclear

power station, biological systems, flow rates, etc. It may in fact be applied to any situation in which a software system is, or may be, of use. It will be appreciated that the screen shots provided in Figures 8 to 30 provide portions of the graphical tool that in this embodiment is used to create and deploy the application.

[0115] In this example as shown in Figure 6, a bespoke system is being devised for a company called HyCar 650, which provides various models and colours of cars. HyCar 650 principally comprises a manufacturer called HyCarManufacturer 652 and a distributor, called HyCarDistributor 654. Together, these two businesses can be thought of as comprising a collaborative network, labelled HyCarNetwork 650.

[0116] HyCarManufacturer 652 provides an Order Processing Service 656 for ordering cars. It also has a Fulfilment Process 658 for fulfilling the order. The fulfilment process validates the order, checks stock availability, schedules production, or raises a shipping note, raises an invoice, and updates the Distributor on progress.

[0117] HyCarDistributor 654 has an eProcurement Service 660 for placing orders to car manufacturers.

[0118] A specification 300 to model this situation is constructed in four levels. The levels are Business Networks, (representing networks of businesses interacting, e.g. a marketplace of buyers and sellers), Businesses, (representing individual businesses), Services (representing the services offered by Businesses) and Processes (representing internal business processes).

[0119] In this example, there is one Business Network- The HyCarNetwork 650. There are two Businesses: HyCarManufacturer 652 and HyCarDistributor 654. HyCarManufacturer 652 offers a Service (the Order

Processing Service) 656 and an internal Process (the Fulfilment Process) 658. HyCarDistributor 654 has a Service (the eProcurement Service) 660.

[0120] A use of the system is now described with reference to the flow chart of Figure 7 and the screenshots of Figures 8 to 30.

[0121] Figure 8 shows a Login screenshot 700, which is displayed on the display means 102, providing a Username entry box 702 and a Password entry box 704 both of which must be filled in to provide access to the system. Such a log in screenshot 700 will be familiar to those in the art.

[0122] It will be appreciated that this screenshot and others yet to be defined are provided by a Graphical User Interface (GUI's). A user of the system uses the keyboard 104 and the mouse 106 to make inputs to the Username entry textbox 702 and the Password entry textbox 704 in step 602. The user thereby accesses the New Project screenshot 800 shown in Figure 9, which comprise a Project Name entry box 802. Again, the user makes an appropriate input to the Project Name entry textbox 802 in step 604. In this example, the project is given the name 'example'.

[0123] The user then progresses to the Design screenshot 900, a section of which is shown in Figure 10.

[0124] The system development occurs in two stages and the first of these is to create patterns 500. These patterns 500 will be cloned to form runtime patterns and will exist within one of the above-defined four layers- Business Networks, Businesses, Services, and Processes. The second stage, as described below, is to create assets accessible to the patterns 500. It will be appreciated from Figure 5 that a pattern is a collection of interconnected nodes 400.

[0125] The Design screenshot 900 comprises a display of the layers of the system and the patterns they are to contain at runtime. At this stage in the example, no patterns have been defined. The Design screenshot 900 further comprises navigation buttons (an Assets button 902, a Patterns button 904, a Runtime Patterns button 906 and a Deployment button 908). These buttons allow a user to navigate from one screen to another by positioning a pointer displayed on the display means 102 and controlled by the mouse 106 over a button and clicking in a manner familiar to those skilled in the art.

[0126] Clicking on the Patterns button 904 takes the user to screens that allow the patterns to be defined using the Pattern Creation means 214 in the Program data portion 150 of the memory 112, 114. In this example, a pattern to be associated with the Services layer is to be defined. A GUI providing a New Pattern screenshot 1000 in Figure 11 allows a new pattern to be defined, in step 602. The New Pattern screenshot 1000 comprises a Pattern Name entry textbox 1002, in to which the user has entered a name for the pattern, which in this example is 'OrderProcessing', intended to provide the Order Processing Service 656 offered by HyCarManufacturer 652.

[0127] The user then progresses to a Pattern Viewer screenshot 1100 as is shown in Figure 12. From this screenshot 1100, the user is able to arranged nodes 400 into patterns 500. To this end, the screenshot 1100 has a Move Node button 1002, an Add Node button 1104, a Delete Node button 1106 and an Add Link button 1108. However, no nodes 400 have yet been defined, so the user clicks the Add Node button 1104 to access the Node Creation means 214 and create a node using a Node Viewer screenshot 1200 shown in Figure 13.

[0128] The Node Viewer screenshot 1200 comprises an Action Name entry box 1202 and an Action list 1204 (at present, no actions have been defined, so the list 1204 is empty). The user creates a new node, called 'Order' and a new

action, labelled 'Sell'. What this action 'Sell' does is yet to be defined but will be defined in rules 304. The new node is stored in the Node storage means 202.

[0129] For the purposes of this example, the pattern is now complete and comprises this one node 400 'Order', arranged to carry out the action 'Sell'. The system uses the input information to build an XML file called 'OrderProcessing.XML' to represent the pattern OrderProcessing. A link between the Order node and the OrderProcessing pattern is created by the linking means 222. This pattern is stored in the Pattern Storage means 204 and is now ready to be 'cloned' by the Pattern Cloning means 220 in step 608 to form a runtime pattern as follows.

[0130] A Runtime pattern is created through clicking the Runtime Patterns button 906 on the Design screenshot 900 shown in Figure 10. This allows a user to navigate to a screen, which is shown as a Runtime Patterns screenshot 1300 in Figure 14. The screenshot 1300 provides an interface that allows the user to create a new Runtime Pattern called OrderProcessing (named using a runtime pattern name entry box 1302). The screenshot further comprises a Clone Pattern' button 1304. On clicking this button 1304, the user is presented with a screen as shown in Figure 15.

[0131] Figure 15 shows a Clone Base Pattern screenshot 1400. The Clone Base Pattern screenshot 1400 comprises a list 1402 of XML 'base' patterns that have previously been defined as XML files in the manner described above for the OrderProcessing pattern. In fact, as the OrderProcessing.XML pattern is the only base pattern defined, the list 1402 is limited to this alone. The Clone Base Pattern screenshot 1400 further comprises a Clone Pattern button 1404. Once a base pattern has been selected, the pattern is cloned when the user clicks on the Clone Pattern button 1404 to produce a runtime pattern called OrderProcessing.

[0132] When a base pattern is cloned, it creates a Runtime Pattern with the same structure as the base pattern. This means that the Runtime Pattern will contain all the nodes 400, actions (such as the action 'Sell' created above), and interactions defined for the pattern. In other systems, the list 1402 may further comprise generic patterns. These would provide a set of patterns that are likely to specify the best way of doing certain tasks. Cloning also creates associates a default (blank) rule set with the message transceiver called OrderProcessing_Order_rules.XML. This is utilised as described below.

[0133] The Runtime Pattern OrderProcessing may then be added to the blue print using the GUI provided by the Design screenshot 900 shown in Figure 9, a portion 1500 of which is shown in Figure 16.

[0134] In order to complete the system for HyCar, it is considered, in step 610, if further patterns should be created. This is the case here, but these further patterns will not be described as exhaustively as the OrderProcessing Pattern. The skilled person will understand that they could be defined in a similar fashion using the GUI's (or other, similar GUI's) described above. It should be understood that a layer can be concrete or can be abstract. A concrete layer is one that can be implemented within the runtime environment. For example, a process or service, as described above, can be implemented within a concrete layer. Abstract layers provide context for the solution being modelled but cannot contain any runtime information. For example, a Business will never have a physical implementation in this example, but will provide context during modelling for the overall application. The following patterns would be created by the user and stored in the Pattern Storage means 204:

1. Business Network Pattern, from which HyCarNetwork can be modelled. For the purposes of this example, this pattern is called P2PMessagingHub. The pattern is abstract and has no nodes 400. The main processing will be conducted in the Service and Process layers. To do this, the user will click on the Patterns

button 904 on the Design screenshot 900 shown in Figure 9, and follow a 'Networks' link to a further GUI, which allows the name of the network to be entered as a New Filename box.

2. Seller Pattern representing the Business HyCarManufacturer. This pattern is abstract, having no nodes 400 and nothing further has to be done at this stage. This is done as above, with the exception that the user follows links to create a business instead of networks.

3. Buyer Pattern representing the business HyCarDistributor. This pattern is also abstract.

4. eProcurement Pattern representing HyCarDistributor's eProcurement Service. The Pattern has a single Node called OrderPlacement with a single action called buy.

5. OrderProcessing Pattern representing HyCarDistributor's order processing Service, as described above.

6. Fulfilment Pattern, representing Hycarmanufacturers validation process as described above.

[0135] The second stage in the development of the Specification 300 is to create assets for the patterns as defined above to use in step 612. The assets are created using the Asset Creation means 216 and, once created, exist in an 'Assets Pool' in the Asset Storage means 206. These Asset Pools are arranged to allow XML Assets to be categorised, stored, and re-used. The user needs to attach the relevant XML assets to the nodes for each Runtime Pattern. Examples of assets include:

1. Messages: These are definitions of the XML messages that will flow between nodes 400.
2. Schema: Schemas can be used to validate the content of XML messages entering nodes. Schemas provide information and data models that may have been agreed upon by industry consortiums.
3. Stylesheets: In this example, these are XSL Stylesheets. Many e-Business solutions will require transformations of incoming XML data to create the outgoing data. An example of this might be the transformation of a BASDA order request into the internal definition required for an existing order entry application. BASDA stands for Business Application Software Developers Association and is involved in setting international standards in document formats.
4. Rule set: These are XML rule sets and contain the Business Rules used by applications
5. Other Resources: With e-Business solutions there are often situations where other content is required. Some examples of this include configuration files, data-to-XML mappings, and process flow definitions.

[0136] Patterns are associated 613 with one another during development of the components of the application and this is an iterative step that is performed as and when required as the patterns become available.

[0137] The example of the figures continues with the creation of assets for the pattern OrderProcessing (step 614) and more specifically with a Message asset defining the content of a Purchase Order.

[0138] First, the user returns to the designs screenshot 900 shown in Figure 9 and clicks on the Assets button 902. This provides access to the Assets Creation means 216. The user is then presented with an assets categories screenshot 1600 as is shown in Figure 17. The assets categories screenshot 1600 comprises a rule set Category button 1602, a Message Category button 1604 and an Other

Resources Category button 1606. The user clicks on the Message Category button 1604 and is presented with a GUI as shown in the New Message screenshot 1700 in Figure 18.

[0139] The New Message screenshot 1700 comprises a Message Name entry box 1702, and a New Message button 1704. The user inputs the name of the new message into the Message Name entry box 1702, which in this case is 'PO.XML' (PO being an abbreviation of Purchase Order) and clicks the New Message button 1704. The user is then presented with the GUI as shown in a Message Content screenshot 1800 of Figure 19.

[0140] The Message Content screenshot 1800 comprises a Document Content entry box. An example XML purchase order should be copied into this box. An appropriate example is shown in Appendix I. This example will be used as a template to create future purchase orders.

[0141] A further asset is created by clicking on the Other Resources Category button 1606 on the Assets Categories screenshot 1600. This asset is called TermsAndConditions.XML and contains the following XML code:

```
<ts_and_cs>
  <delivery>Goods are delivered within 28 days from date of order, unless specified otherwise</delivery>
  <payment_period unit="days">30</payment_period>
  <refunds_returns>
    Returns will only be accepted if goods are in original condition, including packaging.
    Refunds are only payable if goods are faulty and returned within 28 days of receipt.
  </refunds_returns>
</ts_and_cs>
```

[0142] Now that the assets are in place, some rules should be defined so that the assets and the patterns can interact (step 616). The user therefore proceeds as follows, utilising the Rule Creation means 218.

[0143] From the Designs screenshot 900 shown in Figure 9, the user is able to navigate to a screen setting the run-time associations of the OrderProcessing pattern. A Runtime Associations screenshot 1900 is shown in Figure 20. The Runtime Associations screenshot 1900 comprises an OrderProcessing button 1902 buttons showing the runtime associations (or links provided by the Linking means 222) of the defined OrderProcessing patterns. In this example, there are at present no runtime associations for this pattern. Patterns can however be linked (or 'spliced') using the linking means using this GUI. By clicking on the OrderProcessing button 1902, the user accesses a GUI comprising a Pattern Node Viewer screenshot 2000 as shown in Figure 21.

[0144] The pattern Node Viewer Screenshot 2000 comprises a Node button 2002. The Node button provides a link to the node 400 called Order, defined above. By clicking on the Node button 2002, the user accesses a screen as shown in the Rulemaker screenshot 2100 of Figure 22. The rule being constructed in this example is:

'IF an incoming message has the structure of a purchase order, THEN add Terms and Conditions'

[0145] The Rulemaker Screenshot 2100 identified the runtime pattern 2102 (OrderProcessing.XML), its associated node 2104 (Order) and the associated rule set 2106 (OrderProcessing_Order_rules.XML), which it should be remembered was a blank rule set created on cloning the runtime pattern. The Rulemaker Screenshot 2100 further comprises Rule Name entry box 2108 and an Insert Rule button 2110. The user creates a new rule by typing parse_ts_and_cs into the

Rule Name entry box 2108, and clicking on the Insert Rule button 2110 to add the rule. The user then clicks on the new rule to go through to a further GUI such as is shown in Figure 23 comprising a Rules screenshot 2200.

[0146] A rule consists of two components - the Clause and the Result. The Rules screenshot 2200 shows the GUI for Conditions which govern the Clause component of a rule. A portion of the GUI is shown in a Conditions screenshot 2300 of Figure 24.

[0147] The first stage in defining a new rule is to create a new clause. The Conditions screenshot comprised a Clause button 2302. The user can click on the Clause button 2302 and create a new clause called check_root_element. This creates a check_root_element clause button 2304 on the Conditions screenshot 2300. Clicking on the check_root_element clause button 2304 takes the user to a GUI as shown in a Clause screenshot 2400 of Figure 25.

[0148] The Clause screenshot 2400 comprises a parameter button 2402 allowing a user to set the parameters for a clause.In this case the rule is to be an XPath rule. Xpath, as will be known to those skilled in the art, is a language for addressing parts of an XML document. By clicking on the parameter button 2402, the user accesses a GUI such as is shown in Figure 26, showing a Parameter screenshot 2500. This comprises a Value entry box 2502. A Value, in this context, is the value in the Condition. For example, in the Condition, If X is true, then do Y, X is the Value. The user types in the XPath to the value that we wish to test for - in this case, /po- in the Value entry box 2502. It should be remembered that po was the message asset defined above. The condition is therefore 'IF the incoming message has the format of purchase order'.

[0149] Now that the Condition has been defined, the Result (the Y of the above example) needs to be defined. This is achieved by navigating from the Rulemaker Screenshot 2100 to a Results screenshot 2600 as is shown in

Figure 27. The Results screenshot 2600 comprises a Results Name entry box 2602 and an Insert Rule button 2604. The user proceeds by entering the name 'parse_file' in the Results Name entry box 2602 and clicking the Insert Rule button 2604. The user can then access a Results Detail screenshot 2700 as is shown in Figure 28. As well as summarising the progress to date, the Results Detail screenshot 2700 comprises a drop down box 2702 comprising a list of operators. Examples of operators include parsing, transferring, validating, etc.

[0150] In a similar manner to that described above for the Parameters of the Condition, the Parameters of the Result have to be set. The user defines an Xpath parameter called 'to_location' with a value of /po, and a filechooser parameter called 'document' with the value 'TermsAndConditions.XML, the asset defined above. A filechooser is arranged to allow a user to browse through the assets. The result of this is summarised in Figure 29. The completed rule is then stored in the Rule Storage means 208.

[0151] Next the user must associate the TermsAndConditions.XML file with an message transceiver using the linking means 222 so that it will be deployed with it (step 618). This is done though the clone base pattern screenshot 1400 GUI. By following links, the user reaches a GUI as shown in an Engine and Resource Details Screenshot 2900 of Figure 30. This screenshot 2900 comprises an Available Resource Categories drop down box 2902, from which the user may select a category 2904, in this example, the Other Resources category. The screenshot 2900 further comprises a Resources in Category drop down box 2906, from which the user may select a resource 2908, in this case, TermsAndConditions.XML. Finally, the screenshot 2900 comprises an Add Resource button 2910. The user can add the selected resource by clicking this button 2910. Through this GUI, resources can be linked to or unlinked from, nodes.

[0152] At the time of deployment links between the patterns, nodes, etc. become specific. During the creation of the components links between the components are specified 613. However, at the creation time the links do not relate to the physical arrangement of the system on which the application will be run. Therefore, at deployment time the physical nature of the links is specified 619. This may include specifying the location of the component that the link links (and therefore whether the link is a virtual link on the same processing circuitry or whether the link links two or more processing circuits, etc.), what transmission protocol the link will use (HTTP, SMTP. SOAP, etc.).

[0153] An example of this can be seen in Figure 31. During creation of the application (the subject of this Figure) five nodes 3100, 3102, 3104, 3106, 3108 have been created and are linked as shown in the Figure. During creation of the application these links are not specific and it is only during deployment and in particular step 619 that the links are fully defined. For example, all five of the node 3100-3108 could be mapped to run on the same computer 3110.

[0154] In an alternative mapping, also shown in Figure 31, two of the nodes 3100 and 3102 are mapped to a computer 3112, two of the nodes 3104 and 3106 are mapped to a computer 3116 and the fifth node 3108 is mapped to a PDA 3114. The necessary links to connect the five nodes 3100-3108 are defined during step 619 of the deployment.

[0155] The project is now ready to be deployed in step 620 using the Deployment means 224 which in turn accesses the Deployment platform 210. This is achieved though clicking the Deployment button 908 on the Design screenshot 900.

[0156] From this description, it should be appreciated that the system described herein, being highly modular, is highly adaptable. Once a node or an asset has been established, assets can be associated with nodes and Nodes linked

to each other, and such associations and links changed, with great ease. The same is true for links between patterns. Further, the graphical character of the system makes it easy for a user to keep track of the system as it develops and to identify where changes need to be made.